Detecting Correct Execution of Barbell Lifts

Jeroen van Zundert

August 22, 2015

Data courtesy of:

Ugulino, W.; Cardador, D.; Vega, K.; Velloso, E.; Milidiu, R.; Fuks, H. Wearable Computing: Accelerometers' Data Classification of Body Postures and Movements. Proceedings of 21st Brazilian Symposium on Artificial Intelligence. Advances in Artificial Intelligence - SBIA 2012. In: Lecture Notes in Computer Science. , pp. 52-61. Curitiba, PR: Springer Berlin / Heidelberg, 2012. ISBN 978-3-642-34458-9. DOI: 10.1007/978-3-642-34459-6_6.

Detecting Correct Execution of Barbell Lifts

Synopsis

In this analysis we use data on nearly 20000 barbell lifts. Data contains measurements of accelerometers on the belt, forearm, arm and dumbells. The goal is, given these measurements, to classify the exercises in one of five separate classes. Classification has been done using a simple classification tree and random forests. The random forest method performs very well. The out-of-sample error is estimated to be 0.43% using 2-fold cross-validation. The most important variable is roll_belt.

Data Processing

The data is obtained from the internet and is available in a csv format.

```
local_file <- 'pml-training.csv'
if (!exists(local_file)) {
  download.file('https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv', local_file,'wge
}
training <- read.csv(local_file)

local_file <- 'pml-testing.csv'
if (!exists(local_file)) {
  download.file('https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv', local_file,'wget
}
testing <- read.csv(local_file)
```

Remove NA columns and columns 1 to 7 which do not contain relevant data (such as name of the person, time stamp at which exercise was performed, etc)

```
col_select <- (colSums(is.na(testing))==0) & (colSums(is.na(training))==0)
testing_clean <- testing[, col_select]
testing_clean<-testing_clean[,8:ncol(testing_clean)]

training_clean <- training[ , col_select]
training_clean<-training_clean[,8:ncol(training_clean)]
```

Prediction

We set the seed and use 2-fold cross-validation to assess the out-of-sample error. Two-fold is sufficient with such a large data set (nearly 20000 observations with only 52 predictors). Also, running time becomes a problem otherwise with more elaborate algorithms.

The algorithm for this problem should be able to make a distinction between 5 classes. Classification tree and random forest algorithms are well suited for this. I start with a simple classification tree to set a benchmark that runs fast:

```
library(caret)
```

```
## Loading required package: lattice
## Loading required package: ggplot2
```

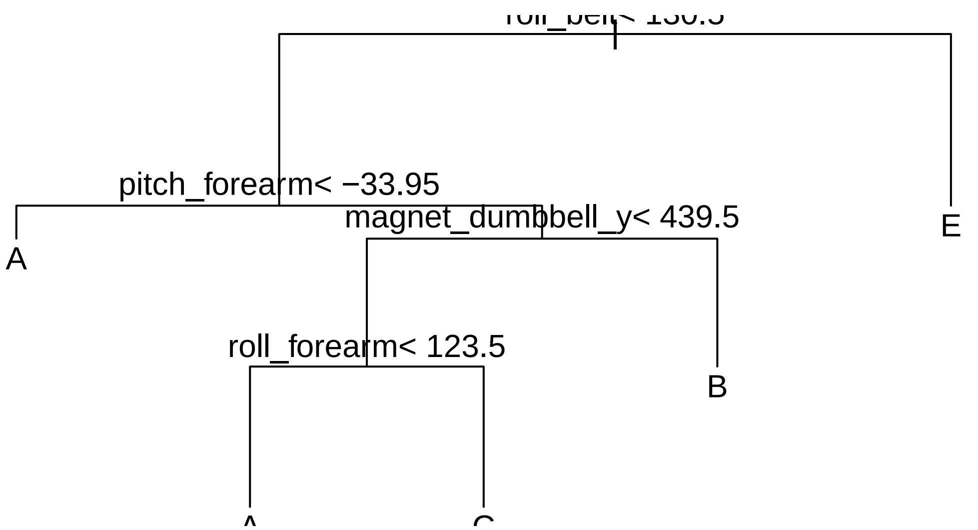
```
set.seed(1000)
fitcontrol <- trainControl(method="cv", number=2) # 2-fold cross validation
fit <- train(classe ~., data=training_clean, trControl=fitcontrol,  method="rpart") # train with rpart
```

```
## Loading required package: rpart
```

```
print(fit)
```

```
## CART
##
## 19622 samples
##    52 predictors
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (2 fold)
## Summary of sample sizes: 9810, 9812
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa       Accuracy SD  Kappa SD
##   0.03567868  0.5231400  0.38861731  0.03704878   0.06499240
##   0.05998671  0.4308872  0.23294913  0.09341638   0.15515565
##   0.11515454  0.3245888  0.06161876  0.05691227   0.08714209
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was cp = 0.03567868.
```

```
plot(fit$finalModel)
text(fit$finalModel)
```




```r
table(predict(fit), training_clean$classe)
```

```
##
##        A    B    C    D    E
##   A 5080 1581 1587 1449  524
##   B   81 1286  108  568  486
##   C  405  930 1727 1199  966
##   D    0    0    0    0    0
##   E   14    0    0    0 1631
```

The accuracy is 52.3%. The matrix shows that all but class A are very hard to predict for a straightforward classification tree algorithm.

Next I use the random-forest method:

```r
set.seed(1000)
fitcontrol <- trainControl(method="cv", number=2)
fit <- train(classe ~., data=training_clean, trControl=fitcontrol,  method="rf")
```

```
## Loading required package: randomForest
## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.
```

```r
print(fit)
```

```
## Random Forest
##
## 19622 samples
##    52 predictors
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (2 fold)
## Summary of sample sizes: 9810, 9812
## Resampling results across tuning parameters:
```

```
##
##   mtry  Accuracy   Kappa      Accuracy SD  Kappa SD
##    2    0.9891959  0.9863324  0.001728191  0.002184597
##   27    0.9898075  0.9871060  0.002304862  0.002915395
##   52    0.9844057  0.9802698  0.006196018  0.007842496
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 27.
```

```r
print(fit$finalModel)
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 27
##
##         OOB estimate of  error rate: 0.43%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 5576    3    0    0    1 0.0007168459
## B   19 3773    5    0    0 0.0063207796
## C    0   10 3402   10    0 0.0058445354
## D    0    0   22 3190    4 0.0080845771
## E    0    1    5    5 3596 0.0030496257
```

Which has a near zero eror rate (0.43%). Based on this cross-validated training, the out-of-sample error is expected to be only 0.43%.

What also is interesting are the variables with the most impact:

```r
varImp(fit)
```

```
## rf variable importance
##
##   only 20 most important variables shown (out of 52)
##
##                      Overall
## roll_belt            100.000
## pitch_forearm         58.804
## yaw_belt              53.446
## pitch_belt            43.768
## magnet_dumbbell_y     43.282
## roll_forearm          42.465
## magnet_dumbbell_z     41.985
## accel_dumbbell_y      21.112
## roll_dumbbell         16.871
## accel_forearm_x       16.572
## magnet_dumbbell_x     16.486
## magnet_belt_z         14.801
## total_accel_dumbbell  14.063
## magnet_forearm_z      13.651
```

```
## accel_dumbbell_z      13.560
## accel_belt_z          13.506
## magnet_belt_y         11.331
## yaw_arm               10.929
## gyros_belt_z          10.812
## magnet_belt_x          9.576
```

roll_belt is the most important variable, followed by pitch_forearm and yaw_belt.